Санкт–Петербургский Государственный Университет
Институт Информационных Технологий

Современные средства описания машинных архитектур

Д.Ю.Булычев
Е.Н.Вигдорчик
Д.С.Ломов
М.Н.Смирнов

Санкт–Петербург
2001

Содержание

1 Введение **3**

2 VHDL (VHSIC (Very High Speed Integrated Circuit) Hardware Description Language)) **3**

3 SLDL (System Level Design Language) **4**

4 LISA: язык описания программируемых архитектур **5**

5 nML **6**

6 Транслятор CBC **7**

1 Введение

Данный отчет посвящен обзору средств описания машинных архитектур и порождения по ним средств кодизайна. Описываются следующие средства:

- язык описания архитектуры VHDL
- группа языков SLDL
- язык описания симуляторов LISA
- язык описания архитектуры nML
- компилятор CBC

Кроме этих подходов, существуют еще формализмы MIMOLA и Verilog, однако они по своей идеологии и возможностям в целом аналогичны VHDL.

Следует отметить, что данная область является относительно новой и на данный момент не выработан общий удачный подход к описанию архитектуры вычислительного устройства, который позволял бы автоматически получать все средства кодизайна. Кроме того, большинство новых предложений не подкреплено никакими реализациями (например, SLDL) или эти реализации недоступны для непосредственного изучения (например, LISA, CBC).

2 VHDL (VHSIC (Very High Speed Integrated Circuit) Hardware Description Language))

Язык VHDL [5, 6, 7], предназначенный для описания цифровых устройств, был создан в 1987 году, стандартизован в IEEE и является крайне мощным и популярным средством разработки. VHDL используется крупными корпорациями и компаниями, для него существует много уже написанных симуляторов, компиляторов и т.д. Описание устройства на VHDL может производится на любом уровне детализации: от простой программы на Аде (VHDL создан на основе Ады) до netlist'а.

VHDL имеет модульную структуру с ясно прописанными с помощью сигналов параметризуемыми интерфейсами и связями. Общая структура программы на VHDL такова: вся она поделена на модули, интерфейсами которых являются наборы входных, выходных или входных-выходных сигналов, с помощью которых модули связываются. Интерфейсы модулей, называемые entity, и их тела (architecture) описываются отдельно.

Внутри архитектуры может быть набор процессов, работающих параллельно. Выполнение процесса начинается при изменении значения любого сигнала, прописанного в его интерфейсе. Внутри процесс последователен. Внутри процесса могут быть использованы процедуры и функции. И те, и другие не могут содержать в себе процесс и быть должны быть синхронными.

Особенность разработки устройств на VHDL заключается в том, что для успешного синтеза описание должно быть максимально подробным и выполненным по специальным правилам. В принципе, программа на VHDL может содержать произвольные конструкции. Можно использовать циклы (loop), if, case, goto, процедуры и функции. Синтезирующие средства не способны превратить в netlist разве что только операторы работы с файлами. Однако для того, чтобы эффективность порождения (количество логики, регистров) было минимальным для данной задачи, необходимо придерживаться ряда соглашений:

- делать дизайн максимально синхронным
- по возможности использовать case вместо if
- никаких циклов
- никаких goto

Автоматически извлечь из описания процессора на VHDL что-либо о системе команд и особенностях их выполнения представляется крайне затруднительным. С другой стороны VHDL можно использовать для описания архитектуры, автоматически получаемого из описания системы на каком-либо другом языке.

3 SLDL (System Level Design Language)

Данный язык [1, 2, 3, 4] задумывался как основное средство проектирования устройств. Он должен позволять инженерам описывать одно- и многопроцессорные встроенные системы с любым уровнем детализации, а также позволять инженерам верифицировать и/или обосновывать их с помощью различных средств. При этом языки представления описывают всю разрабатываемую систему, включая програмное обеспечение, сигнальный уровень, оптические и микроэлектронные механические характеристики. Разработкой языка занимается SLDL комитет, который ставит своей целью поддержку создания и/или стандартизацию языка или даже набора языков, которые будут выполнять эти функции.

Основной целью, для которой этот язык разрабатывался, является возможность описать некую систему на уровне требований заказчика. Так как теперь практически нет ни одного продукта, где не смешивались бы программное обеспечение и электроника, то потребовался язык, позволяющий описать такую систему целиком.

Кроме того, описание системы предлагается разделить по уровням представления. Для не-электронщиков вместо описания электронной части предоставляется некое функциональное представление (на некоем абстрактном уровне) и структурный дизайн (также на некоем абстрактном уровне). Для менеджеров предоставляется информация о стоимостях, версиях и целях.

Общие требования к языку, сформулированные в апреле–июле 1997 года, таковы:

- спецификация требований и ограничений
- системный уровень для интерфейсов с компонентами
- формальная семантика

Данные о системе SLDL устанавливает в четырех основных областях:

- требования (цели дизайна, например прирост на усилителе должен быть больше 7db)
- ограничения и условия на систему (требования, накладываемые окружением системы, например ограничение по излучаемой мощности 5 ватт)
- функциональность (связи между элементами системы, например какой входной сигнал рассматривается как носитель заданной частоты)
- структура (устройство системы, например netlist)

Язык подразумевает высокую степень детализации, поддержку всех видов дизайна (top-down, bottom-up, middle-out), высокую степень связности уровней описания системы (для удобства написания настраиваемых компиляторов, синтезирующих и моделирующих средств). Также предполагается обеспечить возможность частичного описания системы с добавлением недостающих для анализа сведений из набора библиотек (заглушки), специфицирование значений и смысловой нагрузки параметризованных величин, наличие шаблонов и т.д. Для некоторых аспектов описания системы может использоваться быстрое прототипирование. Благодаря всему этому предполагается высокая точность описания системы, легкость проведения различного рода анализов и оценок.

Достигнуть всего этого предлагается в два этапа:

1. сосредоточиться на требованиях и ограничениях, добавив языки описания архитектур (VHDL, Verilog и другие) в поток реализации
2. полностью покрыть весь процесс разработки с помощью языка/группы языков SLDL (встроить в него уже имеющиеся языки и связать их для удобства поддержки проекта или разработать новые)

Первую стадию предполагалось закончить к 2000 году, а к концу 2001 уже получить промышленную реализацию. Наибольшая активность в рамках данного проекта относится к 1998 году и семинару по codesign в НАТО. Данный проект пока не поддержан продуктами.

4 LISA: язык описания программируемых архитектур

LISA – язык описания программируемых архитектур – разрабатывается в Institute for Intergrated Signal Processing Systems, Aachen University of Technology, Германия.

Целью его создания является преодоление семантического разрыва между низкоуровневым описанием процессоров (таким, как VHDL и Verilog) и высокоуровневого описания в виде описания системы команд.

Первоначальной целью создания LISA было построение точных симуляторов машинных архитектур. На данный момент обеспечивается построение:

- точных симуляторов архитектуры (разного типа)
- ассемблера и дизассемблера

Ведутся также работы по автоматическому порождению кодогенератора, но об их результатах ничего не известно.

О работах по порождению самой архитектуры процессора по описанию на LISA также ничего не известно.

Описание архитектры на LISA состоит, с одной стороны, из *описаний ресурсов*, таких как память, регистры, шины и т.д., а с другой стороны, из описания *операций*. Операции являются для LISA основной единицей описания архитектуры. Операции могут быть сложными, включающими одна другую, формируя, таким образом, иерархию операций. На вершине этой иерархии находятся инструкции, а также одна операция – главный цикл загрузки-декодирования-исполнения инструкций.

Каждой операции могут быт приписаны атрибуты, определяемые пользователем. Для средств генерации, реализованных в LISA, используются следующие атрибуты:

- CODING – двоичный код, соответствующий операции
- SYNTAX – текстовое (ассемблерное) представление операции
- BEHAVIOR – семантика операции: код на языке С, оперирующий над ресурсами и описывающий порядок выполнения операции
- ACTIVATION – описывает временные метрики данной операции и их зависимость от времен исполнения других операций
- SEMANTICS – в будущем должна описывать семантику опрерации с точки зрения генерации кода

Создатели языка приложили большие усилия для обеспечения удобства описания архитектуры. В LISA имеется развитая модульная система, позволяющая достигнуть высокого переиспользования похожих фрагментов в описании архитектуры.

Основная цель описания архитектуры на LISA – именно построение симулятора архитектуры. Точность симулятора обеспечивается детальностью поведенческой модели системы, и LISA предоставляет весьма мощный инструмент для построения такой модели.

Однако при этом семантика поведения системы не формализована (это код на С), и построение кодогенератора по существующей схеме описания архитектуры представляется проблематичной.

5 nML

Язык описания архитектуры nML [11] создан в Техническом Университете г.Берлина в конце 80-х — начале 90-х годов.

Текст на nML содержит описание всех основных компонентов машинной архитектуры с точки зрения системы команд. Так, он описывает память, стек, регистры, режимы адресации, собственно набор инструкций. При этом каждая инструкция описывается как процедура в С-подобном языке, тело которой содержит описание семантики инструкции, выраженное в терминал операций над регистрами, памятью и т.д.

Такая организация nML-описания делает принципиально возможным автоматическое порождение генератора кода. Ниже рассмотрен один из подходов, использующих эту идею.

Интересной особеностью nML является то, что для описания групп родственных команд использован механизм атрибутных грамматик. Именно, описание инструкции рассматривается как правило в такой грамматике, а некоторые ее параметры — как нетерминалы в правой части. Таким образом, удается описать сразу группу инструкций, каждая из которых отличается от других типами аргументов (например, операндами, режимами адресации, другими признаками). При это каждый член группы получается в результате обычного вывода из данной инструкции.

Кроме того, нетерминалы в грамматике nML–описания снабжены атрибутами. Атрибуты представляют из себя способ передатьнекоторые дополнительные сведения — например, правила кодирования, предсталение в виде ассемблерного текста и т.д. Каждое правило может требовать наличия каких–либо атрибутов у входящих в его состав нетерминалов и содержать правило их преобразования для вычисления аналогичного (или другого) атрибута у выводимого нетерминала, что обеспечивает возможность вычисления различных характеристик системы.

В то же время nML ориентирован в основном на описание именно ситемы команд. В частности, внутренняя логика работы устройства не отражается явно в nML–описании. Невозможно описать паралеллизм различных блоков, тайминг, и, следовательно, невозможно построение аккуратного симулятора.

6 Транслятор CBC

Транслятор CBC создан в Техническом университете г.Берлина [8, 9, 10].

В CBC сделан акцент на использование сложных потоков данных, практически всегда в изобилии присутствующих в архитектуре DSP вычислительного устройства. Кроме того, разработчики этого инструмента попытались учесть наличие нескольких функциональных блоков в процессоре, работающих параллельно, а также возможность разной ширины шин, и, соответственно, необходимость отображения базовых типов языка высокого уровня в типы, определяемые процессором.

Коренное отличие CBC от традиционных трансляторов кроется во входном языке, на котором описывается целевой алгоритм. Разработчики CBC (вместе с компанией Siemens, поддерживавшей этот проект) сконцентрировали свои усилия на генерации кода для языка AlDisp. Описание алгоритма на этом языке подразумевает графическое или текстуальное задание графа потока сигналов. Фактически это означает задание потока данных с дополненительными конструкциями управления. Таким образом генерация в гораздо большей степени ориентированна на поток данных, а не управления.

Исходя из вышесказанного разработчики CBC решили отказаться от традиционного в компиляторах промежуточного представления программы в виде дерева или ациклического направленного графа (DAG). Вместо этого ими использован смешанный граф потока данных/управления (CDFG). В этом графе вершины представляют операции программы, а дуги либо зависимость по данным, либо дополнительную зависимость по управлению. При этом второй тип дуг является динамическим, то есть такие дуги вставляются собственно генератором кода во время работы, и являются лишь ограничениями, необходимыми для правильной кодогенерации.

Именно чрезмерная ориентация CBC на поток данных, выражающаяся и в языке описания программы, сужает область его применения до небольшого числа классических алгоритмов цифровой обработки данных, в которых поток управления действительно играет второстепенную роль.

Список литературы

[1] Carlos Delgaro Cloos, Simon Pickin, Luis Sanches, Angel Groba. High-Level Specification Languages for Embedded System Design

[2] S.Schulz. SLDL graphics. 1998

[3] S.Schulz. Towards A new System Level Design Language. NATO Summer Institute on System Level Synthesis, 1998

[4] David L. Barton. Systems Level Design Section of the Industry Standarts Roadmap. SLDL Committee, 1998

[5] Mentor Graphics VHDL Reference Manual, 1994

[6] Mentor Graphics Introduction to VHDL

[7] IEEE Standart VHDL Language Reference Manual, IEEE Std 1076-1987

[8] Andreas Fauth, Günter Hommel, Alois Knoll, Carsten Müller Global Code Selection for Directed Acyclic Graphs. Int. Conf. on Compiler Construction CC-94, Edinburgh, Scotland, UK, April, 1994, published in LNCS 768, Springer, pp. 128-142

[9] Andreas Fauth, Alois Knoll. Translating Signal Flowcharts into Microcode for Custom Digital Signal Processors. Proc. Int. Conf. on Signal Processing ICSP-93, Beijing, PR China, October,1993, pp. 65-72

[10] C. Müller. Code Selection from Directed Acyclic Graphs in the Context of Domain Specific Digital Signal Processors. Jahresarbeit, Humboldt-Universität zu Berlin, August 1994

[11] M. Freericks.The nML Machine Description Formalism - Version 2.0. http://www.techfak.uni-bielefeld.de/ags/ti/publications.html